COMPLEX NUMBERS AND FUNCTIONS

1. Basic definitions.

The set of real numbers **R**, together with the algebraic operations of addition and multiplication, is the main example of an algebraic system known as a "field." The quadratic equation $x^2 + 1 = 0$ has no real solution, since $x \in \mathbf{R} \Rightarrow x^2 + 1 \geq 0 + 1 = 1 > 0$.

We *extend* the real numbers **R** to the "field" **C** of complex numbers. Suppose there exists a number i, an *imaginary unit*, not a real number, which solves $z^2 + 1 = 0$. Thus

$$i^2 = -1.$$

Engineers use j for i and matlab responds to either.

The set of *complex numbers* **C** consists of all numbers of the form

$$z = x + iy, \qquad x, y \in \mathbf{R}.$$

We define the *arithmetic operations* with complex numbers and some related concepts. The phrase "a : = b" is read "a is defined to be b," and the phrase "a = : b" means "b is defined to be a." Let

$$z = x + iy,$$

$$z_0 = x_0 + iy_0, \qquad z_1 = x_1 + iy_1,$$

be complex numbers. Then:

a. z_0 and z_1 are *equal* if they are the *same* complex number, that is if

$$\boxed{x_0 = x_1 \quad \text{and} \quad y_0 = y_1.}$$

We then write

$$\boxed{z_0 = z_1}.$$

b. The *sum* of z_0 and z_1 is

$$\boxed{\begin{aligned} z_0 + z_1 &= (x_0 + iy_0) + (x_1 + iy_1) \\ &:= (x_0 + x_1) + i(y_0 + y_1). \end{aligned}}$$

c. The *product* of z_0 and z_1 is defined via the formalism

$$z_0z_1 := (x_0+iy_0)(x_1+iy_1)$$
$$= x_0x_1 + ix_0y_1 + iy_0x_1 + \underbrace{i^2}_{-1} y_0y_1 \,.$$

More precisely,

$$\boxed{\begin{aligned} z_0z_1 &= (x_0+iy_0)(x_1+iy_1) \\ &:= (x_0x_1 - y_0y_1) + i(x_0y_1 + y_0x_1) \,. \end{aligned}}$$

d.

$$\begin{aligned} \mathrm{Re}\, z &:= \textit{real part} \text{ of } z \\ &:= x \\ &= \mathrm{real}\,(z) \quad \text{in matlab.} \end{aligned}$$

$$\begin{aligned} \mathrm{Im}\, z &:= \textit{imaginary part} \text{ of } z \\ &:= y \\ &= \mathrm{imag}\,(z) \quad \text{in matlab.} \end{aligned}$$

e.

$$\begin{aligned} \bar{z} &:= \textit{conjugate} \text{ of } z \\ &:= x - iy := x + i(-y) \\ &= \mathrm{conj}\,(z) = z' \quad \text{in matlab} \end{aligned}$$

f.

$$\begin{aligned} |z| &:= \textit{modulus} \text{ of } z \\ &:= \textit{absolute value} \text{ of } z \\ &:= \sqrt{x^2+y^2} \\ &= \mathrm{abs}\,(z) \quad \text{in matlab.} \end{aligned}$$

Note that addition and multiplication of complex numbers are *commutative*:

$$z_0 + z_1 \equiv z_1 + z_0, \qquad z_0z_1 \equiv z_1z_0 \,.$$

Here, for instance, the phrase "$z_0z_1 \equiv z_1z_0$" is short for "$z_0z_1 = z_1z_0$, *identically*, for *all* $z_0, z_1 \in \mathbf{C}$." To prove this statement, first swap subscripts to get

$$z_1z_0 = (x_1x_0 - y_1y_0) + i(x_1y_0 + y_1x_0) \,.$$

Now use the commutativity of multiplication and addition of *real* numbers to get

$$\begin{aligned} z_1z_0 &= (x_0x_1 - y_0y_1) + i(y_0x_1 + x_0y_1) \\ &= (x_0x_1 - y_0y_1) + i(x_0y_1 + y_0x_1) \\ &= z_0z_1. \end{aligned}$$

The complex number $z = x + iy$ is a "real complex number" if $y = 0$. Otherwise it is *nonreal.* We write

$$x + i0 =: x\,.$$

If $z_0 = x_0$ and $z_1 = x_1$ are "real complex numbers" then so are

$$z_0 + z_1 = x_0 + x_1 \quad \text{and} \quad z_0 z_1 = x_0 x_1.$$

The arithmetic operations with "real complex numbers" obey the same rules as for the real numbers. Thus we may *identify* the "real complex numbers" with the real numbers $\mathbb{R}$. In this way the "field" of real numbers becomes a subfield of the "field" of complex numbers.

The matlab command is real(z) delivers 1 (true) if z is real and 0 (false) if z is nonreal.

If $c \in \mathbb{R}$ and $z = x + iy \in \mathbb{C}$, then

$$\begin{aligned} cz &:= (c + i0)(x + iy) \\ &= (cx - 0y) + i(0x + cy) \\ &= (cx) + i(cy) \\ &= (xc) + i(yc) \\ &= zc\,, \end{aligned}$$

and if $c \neq 0$ then

$$\begin{aligned} \tfrac{z}{c} &:= z \cdot \tfrac{1}{c} \\ &= (x + iy)\left(\tfrac{1}{c} + i0\right) \\ &= \left(\tfrac{x}{c} - y \cdot 0\right) + i\left(\tfrac{y}{c} + x \cdot 0\right) \\ &= \left(\tfrac{x}{c}\right) + i\left(\tfrac{y}{c}\right) \\ &= \left(\tfrac{1}{c}\right) z\,. \end{aligned}$$

The complex number $x + iy$ is *imaginary* if $x = 0$. We write

$$0 + iy =: iy\,.$$

When *numerical* values are given for x and y it is customary to write $x + iy$ as $x + yi$. Matlab does this. In any case either notation is acceptable.

We also put

$$x + i := x + 1i\,.$$

In particular,

$$i := 0 + 1i$$

so

$$\begin{aligned} i^2 &:= i \cdot i = (0+1i)(0+1i) \\ &:= (0 \cdot 0 - 1 \cdot 1) + i(0 \cdot 1 + 1 \cdot 0) \\ &= -1 + 0i \\ &= -1 , \end{aligned}$$

as promised.

The only complex number which is both real and imaginary is the *additive identity element*

$$0 := 0 + 0i .$$

We have

$$z + 0 \equiv z \equiv 0 + z$$

(identically for *all* $z \in \mathbf{C}$).

If $a \in \mathbf{C}$, then the equation

$$a + z = 0$$

has the unique solution

$$\begin{aligned} z = -a &:= -\mathrm{Re}\ a - i\,\mathrm{Im}\ a \\ &:= (-\mathrm{Re}\ a) + i(-\mathrm{Im}\ a) , \end{aligned}$$

the *additive inverse* of a. This can be seen by equating real and imaginary parts:

$$a + z = 0 \quad \Leftrightarrow \quad \mathrm{Re}\ a + \mathrm{Re}\ z = 0, \quad \mathrm{Im}\ a + \mathrm{Im}\ z = 0 .$$

The *multiplicative identity element* is

$$1 := 1 + 0i .$$

We have

$$z \cdot 1 \equiv z \equiv 1 \cdot z .$$

Note that in general

$$\begin{aligned} z\bar{z} &= (x+iy)(x-iy) \\ &= x^2 + y^2 \\ &= |z|^2 \\ &> 0 \quad \text{if and only if } z \neq 0 . \end{aligned}$$

If $a \in \mathbf{C}$ with $a \neq 0$, then the equation

$$az = 1$$

has the unique solution

$$z = \frac{1}{a} = \frac{\bar{a}}{|a|^2},$$

the *multiplicative inverse* of a. We treat the more general case of complex division, carefully from a computational point of view, in the next section. But the formula is easily remembered by the formalism

$$\boxed{\frac{1}{a} = \frac{1}{a}\frac{\bar{a}}{\bar{a}} = \frac{\bar{a}}{|a|^2}}.$$

More generally,

$$\boxed{\frac{a}{b} = \frac{a}{b}\frac{\bar{b}}{\bar{b}} = \frac{a\bar{b}}{|b|^2} \quad \text{if } b \neq 0}.$$

Examples.

1. $\frac{1}{1+i} = \frac{1}{1+i}\frac{1-i}{1-i} = \frac{1-i}{1^2+1^2} = \frac{1}{2} - \frac{1}{2}i\,.$

2. $\frac{1+i}{1-i} = \frac{1+i}{1-i}\frac{1+i}{1+i} = \frac{(1-1)+(1+1)i}{1^2+1^2} = i\,.$

3. $\frac{1+2i}{1+i} = \frac{1+2i}{1+i}\frac{1-i}{1-i} = \frac{(1+2)+(2-1)i}{1^2+1^2} = \frac{3+i}{2} = \frac{3}{2} + \frac{1}{2}i\,.$

Observe that $|z| = 1 \;\Leftrightarrow\; 1 = |z|^2 = z\bar{z}$. Thus

$$\boxed{|z| = 1 \;\Leftrightarrow\; \frac{1}{z} = \bar{z}}.$$

We have shown, mathematically, how to compute our first non-trivial complex function $f(z) = \frac{1}{z}$. We have

$$f(z) = \frac{\bar{z}}{|z|^2}, \quad z \neq 0$$

$$= \infty, \quad z = 0\,.$$

2. Complex division by Gauss factorization.

- We show how to compute the *quotient*

$$w := \frac{z_0}{z_1} \quad (z_1 \neq 0)\,.$$

This is equivalent with solving the equation

$$z_1 w = z_0\,,$$

that is

$$(x_1 + iy_1)(u + iv) = x_0 + iy_0\,,$$

for $w = u + iv$. Equate real and imaginary parts to get

$$x_1 u - y_1 v = x_0\,,$$

$$y_1 u + x_1 v = y_0\,,$$

that is,

$$\begin{bmatrix} x_1 & -y_1 \\ y_1 & x_1 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix},$$

a real linear system of two equations in two unknowns and a *very special matrix*. One way to solve this system is via Gauss factorization with complete pivoting. We don't recommend partial pivoting as a general strategy but here, because of the special form of the matrix, the two pivot strategies are the same.

If $|x_1| \geq |y_1|$ the pivot is already in place and we factor

$$\begin{bmatrix} x_1 & -y_1 \\ y_1 & x_1 \end{bmatrix} = \begin{bmatrix} 1 & \\ \ell & 1 \end{bmatrix}\begin{bmatrix} x_1 & -y_1 \\ & g \end{bmatrix}.$$

This requires

$$y_1 = \ell x_1, \qquad x_1 = -\ell y_1 + g\,.$$

Thus we compute

$$\boxed{\ell = \frac{y_1}{x_1}, \qquad g = x_1 + \ell y_1}\,.$$

Thus we must now solve

$$\begin{bmatrix} 1 & \\ \ell & 1 \end{bmatrix}\begin{bmatrix} x_1 & -y_1 \\ & g \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

for u and v. But, as is easily checked, we have

$$\begin{bmatrix} 1 & \\ -\ell & 1 \end{bmatrix}\begin{bmatrix} 1 & \\ \ell & 1 \end{bmatrix}=\begin{bmatrix} 1 & \\ & 1 \end{bmatrix}=\begin{bmatrix} 1 & \\ \ell & 1 \end{bmatrix}\begin{bmatrix} 1 & \\ -\ell & 1 \end{bmatrix}$$

so this linear system is *equivalent* with the upper triangular system

$$\begin{bmatrix} x_1 & -y_1 \\ & g \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}=\begin{bmatrix} 1 & \\ -\ell & 1 \end{bmatrix}\begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

$$=\begin{bmatrix} x_0 \\ -\ell x_0+y_0 \end{bmatrix}.$$

We now *backsolve* for v and u:

$$\boxed{v=\frac{y_0-\ell x_0}{g}, \qquad u=\frac{x_0+y_1 v}{x_1}}.$$

If $|x_1|<|y_1|$ then we swap the two equations to get

$$\begin{bmatrix} y_1 & x_1 \\ x_1 & -y_1 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}=\begin{bmatrix} y_0 \\ x_0 \end{bmatrix},$$

that is,

$$\begin{bmatrix} y_1 & -x_1 \\ x_1 & y_1 \end{bmatrix}\begin{bmatrix} u \\ -v \end{bmatrix}=\begin{bmatrix} y_0 \\ x_0 \end{bmatrix}.$$

This is like our original system but with the x_k and y_k ($k=0, 1$), and v and $-v$, interchanged.

Thus if $|x_1|<|y_1|$ we compute

$$\boxed{\ell=\frac{x_1}{y_1}, \qquad g=\ell x_1+y_1}$$

and then

$$\boxed{v=\frac{\ell y_0-x_0}{g}, \qquad u=\frac{y_0-x_1 v}{y_1}}.$$

The flop count for this algorithm is

$$3\delta+3\mu+3\alpha+1\gamma=6\mu+3\alpha+1\gamma=10\phi$$

where δ denotes divisions, μ multiplications or multiplicative operations ($\mu \Leftrightarrow \delta$), α additive operations, γ comparisons and ϕ flops (floating point operations). We have described our code *cquotient* for complex division.

Of course in the first case say, for $|x_1| \geq |y_1|$, we have

$$\ell = \frac{y_1}{x_1},$$

$$g = x_1 + \ell y_1 = x_1 + \frac{y_1^2}{x_1} = \frac{x_1^2 + y_1^2}{x_1},$$

$$v = \frac{y_0 - \ell x_0}{g} = \frac{y_0 - \frac{y_1 x_0}{x_1}}{\frac{x_1^2 + y_1^2}{x_1}},$$

$$\boxed{v = \frac{x_1 y_0 - y_1 x_0}{x_1^2 + y_1^2}},$$

and

$$u = \frac{x_0 + y_1 v}{x_1} = \frac{x_0 + y_1 \frac{x_1 y_0 - y_1 x_0}{x_1^2 + y_1^2}}{x_1}$$

$$= \frac{x_0 x_1^2 + x_0 y_1^2 + x_1 y_0 y_1 - x_0 y_1^2}{x_1 \left(x_1^2 + y_1^2\right)},$$

$$\boxed{u = \frac{x_0 x_1 + y_0 y_1}{x_1^2 + y_1^2}}.$$

The final formulas for u and v are the same as those obtained from

$$w = u + iv = \frac{z_0}{z_1} = \frac{z_0}{z_1}\frac{\bar{z}_1}{\bar{z}_1} = \frac{z_0 \bar{z}_1}{|z_1|^2}$$

These formulas cannot be used directly, numerically, because computation of the squares, x_1^2 and/or y_1^2, can cause artificial overflow or underflow and ruin the algorithm.

Smith's algorithm, our code *cdiv*, is somewhat like ours. It computes u and v as follows:

if $|x_1| \geq |y_1|$

$$\ell = y_1/x_1, \quad d = x_1 + \ell y_1$$

$$u = \frac{x_0 + \ell y_0}{d}, \quad v = \frac{y_0 - \ell x_0}{d}$$

else

$$\ell = x_1/y_1, \quad d = \ell x_1 + y_1,$$

$$u = \frac{\ell x_0 + y_0}{d}, \quad v = \frac{\ell y_0 - x_0}{d}$$

end

We merely divide the numerators and denominators by x_1 or y_1, whichever absolute value is larger, and do the computations "wisely." Note that $d = g$! This uses the same work as our algorithm. It seems a slight bit more elegant. But neither is "perfect"!

Besides the obvious practical importance of this section we observe that the real 2-vector

$$z := \begin{bmatrix} x \\ y \end{bmatrix}$$

and special real 2×2 matrix

$$A_z := \begin{bmatrix} x & -y \\ y & x \end{bmatrix}$$

associated with the complex number

$$z = x + iy$$

arose in a natural way.

3. Computing $w = \sqrt{z}$, $z \in \mathbf{C}$.

We put $z = x + iy$ and $w = u + iv$. We want

$$w^2 = z\,,$$

that is,

$$(u + iv)^2 = x + iy\,,$$

that is, equating real and imaginary parts,

$$u^2 - v^2 = x, \quad 2uv = y\,.$$

If w is a solution so is $-w$. We insist that $u \geq 0$. *If* $y = 0$ we take

$$\boxed{\begin{aligned} w = u + iv &:= \sqrt{x}, & x \geq 0\,, \\ &:= i\sqrt{|x|}, & x \leq 0\,. \end{aligned}}$$

Suppose $y \neq 0$. We may then take $u > 0$.

It is convenient to scale by 2. Since IEEE arithmetic is binary this causes no rounding errors. Thus let

$$\boxed{\xi := \frac{x}{2}, \quad \eta := \frac{y}{2} \quad \zeta := \xi + i\eta}\,.$$

Then our equations are

$$u^2 - v^2 = 2\xi, \quad uv = \eta\,.$$

Eliminate $v = \frac{\eta}{u}$:

$$u^2 - \frac{\eta^2}{u^2} = 2\xi\,,$$

$$u^4 - 2\xi u^2 - \eta^2 = 0\,.$$

Solve for u^2:

$$u^2 = \frac{2\xi \pm \sqrt{4\xi^2 + 4\eta^2}}{2}$$

$$= \xi \pm \sqrt{\xi^2 + \eta^2}$$

$$= \xi \pm |\zeta|\,.$$

Since $\eta \neq 0$ we must have the upper sign:

$$\boxed{u = \sqrt{|\zeta| + \xi}, \quad v = \frac{\eta}{u}}\,.$$

This is a *mathematical solution* of our problem. But it is unsatisfactory numerically when $\xi < 0$ because of *cancellation*. But then

$$|\zeta| + \xi = (\sqrt{\xi^2 + \eta^2} + \xi)\frac{\sqrt{\xi^2 + \eta^2} - \xi}{\sqrt{\xi^2 + \eta^2} - \xi}$$

$$= \frac{\eta^2}{|\zeta| + |\xi|},$$

$$u = \frac{|\eta|}{\sqrt{|\zeta| + |\xi|}}$$

and

$$\boxed{v = \operatorname{sign} \eta \sqrt{|\zeta| + |\xi|}, \quad u = \frac{\eta}{v}}\,.$$

Now, to protect against artificial overflow and underflow, $|\zeta|$ is normally computed as

$$\begin{aligned} |\zeta| &= |\xi|\sqrt{1+\left(\frac{\eta}{\xi}\right)^2}, \qquad |\xi|\geq|\eta| \\ &= |\eta|\sqrt{1+\left(\frac{\xi}{\eta}\right)^2}, \qquad |\xi|<|\eta|. \end{aligned}$$

This means that

$$s := |\zeta| + |\xi|$$

should be computed as

$$\boxed{\begin{aligned} s &= |\xi|\left(1+\sqrt{1+\left(\frac{\eta}{\xi}\right)^2}\right), \qquad |\xi|\geq|\eta| \\ &= |\eta|\left(\left|\frac{\xi}{\eta}\right|+\sqrt{1+\left(\frac{\xi}{\eta}\right)^2}\right), \qquad |\xi|<|\eta|. \end{aligned}}$$

The *stable* algorithm is then completed by

```
t = sqrt(s)
if ξ > 0
    u = t,    v = η/t
else
    u = |η|/t,    v = t sign η
end
```

This was a description of our matlab code *csqrt*.

4. Matlab Codes and Diary

function z = cquotient (z0, z1)

z = z0/z1 is the *quotient* of the complex numbers z0 and z1. z is computed stably using Gauss factorization with complete pivoting followed by forward and back solution. See also cdiv.

Algorithms cdiv and cquotient require the same work. They are also quite close with respect to accuracy. They are both slightly better, in this regard, than matlab's complex division. Which one to choose is a (not extremely important) open question.

cquotient calls no extrinsic functions.

```
begin cquotient
    a = real(z1);   b = imag(z1);   u = real(z0);   v = imag(z0);
    if abs(a) < abs(b)
        ℓ = a/b;   g = ℓ*a + b;   y = (ℓ*v - u)/g;   x = (v - a*y)/b;
    else
        ℓ = b/a;   g = a + ℓ*b;   y = (v - ℓ*u)/g;   x = (u + b*y)/a;
    end
    z = x + i*y;
end cquotient
```

Total flops: 3 adds + 6 mults + 1 comparison of absolute values of real numbers.

function w = cdiv(z0, z1)

w = z0/z1 is the *quotient* of the complex numbers z0 and z1. w is computed stably, roughly as matlab should be computing it.

 Revised 15 March 1993.

cdiv calls no extrinsic functions.

```
begin cdiv
    a = real(z0);   b = imag(z0);   x = real(z1);   y = imag(z1);
    if abs(x) < abs(y)
        t = x/y;   d = x*t + y;   u = (a*t +b)/d;   v = (b*t - a)/d;
    else
        t = y/x;   d = x +y*t;   u = (a + b*t)/d;   v = (b - a*t)/d;
    end
    w = u + i*v;
end cdiv
```

Total flops: 3 adds + 6 mults + 1 comparison of absolute values of real numbers.

Remarks:

The algorithm is surely straightforward. Thus, if it is to be named after someone, perhaps "Smith" is appropriate. But there seems to be no reference to any paper by Smith. The algorithm is given as an exercise in Knuth [2]. The algorithm was publicized in [1], an expository paper about what every person who uses it should know about floating point arithmetic.

Added 10 January 1996. Smith's algorithm was actually published in [3]. See also [4] for remarks on the algorithm.

References:

[1] David Goldberg, What every computer scientist should know about floating-point arithmetic. ACM Computing Surveys 23 (1991) 5-48.

[2] D. Knuth, The Art of Computer Programming, Volume 2. Addison-Wesley, Reading, MA, 1969.

[3] Robert L. Smith, Algorithm 116: Complex division. Comm. ACM 5 (1962) 435.

[4] G.W. Stewart, A note on complex division. ACM Trans. Math. Software 11 (1985) 238-241.

Diary cdiv. Computation of quotients of complex numbers in three different ways, on an IBM type PC and an HP work station.

1. On the PC, with matlab 4.2.

```
z0 = rand(10000,1) + i*rand(10000,1);
z1 = rand(10000,1) + i*rand(10000,1);
wq = zeros(10000,1);      wd = zeros(10000,1);
for k = 1:10000 wq(k) = cquotient(z0(k),z1(k));     end
for k = 1:10000 wd(k) = cdiv(z0(k),z1(k));     end
wpc = z0./z1;         % We don't know what matlab does here!
eps = machprec        % This is my, and Lapack's, machine precision.  It is half of matlab's
eps =                 % eps.
    1.110223024625156e-016
[w ww] = div1(z0,z1);     % The "true" quotients computed with sep (simulated extended precision)
                          % arithmetic.
eq = sub21(w,ww,wq)./w;     eq = max(abs(eq))/eps    % The rounding errors computed
eq =                                                 % with doubled precision.
   2.96488082450659
ed = sub21(w,ww,wd)./w;     ed = max(abs(ed))/eps
ed =
   2.74114661503672       % cdiv is slightly better than cquotient.
epc = sub21(w,ww,wpc)./w;     epc = max(abs(epc))/eps
epc =
    0.97517967678905      % Substantially better than our algorithms, but this is because the
                          % Intel-matlab combination uses arithmetic that is frequently better,
diary off                 % but sometimes only slightly worse, than ideal IEEE.  When doing the
                          % division it (presumably) uses a code like one of ours, but it uses an
                          % extended precision to do the arithmetic until it finally rounds the
                          % results to working precision to store them.  So this is not a fair test
                          % for our algorithms.  It would be better if all machines did their
                          % arithmetic the same!
```

2. On the HP, with matlab 4.0a.

```
>> z0 = rand(10000,1) + i*rand(10000,1);     % different random numbers!
>> z1 = rand(10000,1) + i*rand(10000,1);
>> wq = zeros(10000,1);     wd = zeros(10000,1);
>> for k = 1:10000   wq(k) = cquotient(z0(k),z1(k));     end
>> for k = 1:10000   wd(k) = cdiv(z0(k),z1(k));     end
>> whp = z0./z1;
>> eps
eps =
     1.1102e-16
>> [w  ww] = div1(z0,z1);
>> eq = sub21(w,ww,wq)./w;     eq = max(abs(eq))/eps
eq =
     2.7890
>> ed = sub21(w,ww,wd)./w;     ed = max(abs(ed))/eps
ed =
     2.7340      % cdiv is again very slightly better than cquotient.
>> ehp = sub21(w, ww,whp)./w;     ehp = max(abs(ehp))/eps
ehp =
     4.0288
```

The HPs do ideal IEEE arithmetic. Our algorithms are *better* than whatever algorithm matlab is using (again, they don't tell us!). QED (and so ends the demonstration).

Such experiments are *no proof* that the algorithm does its job. That involves *rounding error analysis (not hard) and the presumption* that the machine does its basic arithmetic correctly (cf the recent "pentium fiasco"). Again, *all* floating point arithmetic should be the *same, ideal* IEEE *arithmetic.*

function w = csqrt(z)

w is the principal branch of the *square root* of the *complex* number z, computed stably. We have Re(w) > 0 unless z is zero or a negative real number in which case w = iv with v >= 0.

 Revised 18 March 1993.

csqrt calls no extrinsic functions.

```
begin csqrt
    x = real(z);    y = imag(z);
    if y == 0
        if x > 0
            u = sqrt(x);    v = 0;
        else
            u = 0;    v = sqrt(-x);
        end
    else
        signx = sign(x);    x = abs(x)/2;
        signy = sign(y);    y = abs(y)/2;
        if x > y
            r = y/x;    s = x*(1 + sqrt(1 + r*r));
        else
            r = x/y;    s = y*(r + sqrt(1 + r*r));
        end
        t = sqrt(s);
        if signx > 0
            u = t;    v = y/t;
        else
            u = y/t;    v = t;
        end
        v = v*signy;
    end
    w = u + i*v;
end csqrt
```

Total flops: 2 sqrts + 2 divs + 2 mults + 2 adds + 2 comps.

Problem Set C

1. Let $z_0 = 1 + 2i$ and $z_1 = 1 - i$.

   a. Compute $\operatorname{Re} z_0$, $\operatorname{Re} z_1$, $\operatorname{Im} z_0$, $\operatorname{Im} z_1$, $\bar{z}_0$, $\bar{z}_1$, $|z_0|$ and $|z_1|$.

   b. Compute $z_0 + z_1$, $z_0 - z_1$, $z_0 z_1$, $\frac{1}{z_0}$, $\frac{1}{z_1}$, $\frac{z_0}{z_1}$, $\frac{z_1}{z_0}$, $z_0 \cdot \left(\frac{1}{z_1}\right)$ and $z_1 \cdot \left(\frac{1}{z_0}\right)$.

   [Note: $z_0 - z_1 := z_0 + (-z_1)$.]

2. Show that

$$\overline{a+b} \equiv \bar{a} + \bar{b}, \qquad \overline{ab} \equiv \bar{a}\,\bar{b}$$

(identically, for *all* $a, b \in C$).

3. The *powers* of $z \in C$ are defined by

$$z^n := \underbrace{z \cdot z \cdots z}_{n \text{ times}}, \qquad n = 0, 1, 2, \ldots .$$

   a. Compute i^n, $n = 0, 1, 2, 3, 4, \ldots$.

   b. Show that

$$(-1)^2 = \left(-\frac{1}{2} + \frac{\sqrt{3}}{2} i\right)^3 = i^4 = 1.$$

4. Let

$$w := \frac{1}{\sqrt{5}+1} + \frac{i}{2}\sqrt{\frac{5+\sqrt{5}}{2}}.$$

Show that:

   a. $|w| = 1$,

   b. $w^2 = -\frac{\sqrt{5}+1}{4} + i\,\frac{\sqrt{\frac{5+\sqrt{5}}{2}}}{\sqrt{5}+1}$,

   c. $w^3 = \overline{w^2}$,

   d. $w^4 = \bar{w}$,

   e. $w^5 = 1$.

5. Consider the complex $n \times n$ linear system

$$Cz = c \qquad (*)$$

with

$$C = A + iB, \qquad c = a + ib, \qquad z = x + iy.$$

Suppose you can't find any codes for solving complex linear systems. Show that (*) is equivalent with the real linear system

$$\begin{bmatrix} A & -B \\ B & A \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} \qquad (**)$$

(This uses *block multiplication.*)

6. (Continuation) The flop count for solving an $n \times n$ linear system is about $\frac{n^3}{3}(\mu_F + \alpha_F)$. So the count for solving (*) is about $\frac{n^3}{3}(\mu_C + \alpha_C)$ with the subscripts C denoting *complex* operations. Show that this is $\frac{4n^3}{3}(\mu_R + \alpha_R)$ with the subscript R denoting *real* operations. How much do we *lose* by solving (**) in real arithmetic?

7. Show that (see page 9):

   a. $z = A_z e_1$ with $e_1 := \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ the first axis vector.

   b. $\det A_z = |z|^2 = \|z\|^2$ (square of the Euclidean norm of z).

   c. $A_{z_0} + A_{z_1} = A_{z_0+z_1}, \qquad A_{z_0} A_{z_1} = A_{z_0 z_1}$

   d. $A_{\bar{z}} = A_z'$ (the transpose of A_z).

   e. $A_z' A_z = A_z A_z' = |z|^2 I_2 := |z|^2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

8. A *quaternion* can be described as a matrix of the form

$$Q = \begin{bmatrix} a & -\bar{b} \\ b & \bar{a} \end{bmatrix}, \quad a, b \in \mathbb{C}.$$

Show that the sum and product of two quaternions are again quaternions and that *quaternions do not commute* under multiplication.

The point of 7c is that the *real* quaternions are just a disguised form of the complex numbers. The quaternions can be generalized further, to "Cayley numbers." We *use* (complex) quaternions with $|a|^2 + |b|^2 = 1$ in computational linear algebra; they are *complex rotations.*

9. Compute $\sqrt{i}$ and $\sqrt{-\frac{1}{2} + \frac{\sqrt{3}}{2} i}$.

PERMUTATION MATRICES

The $n \times n$ *identity matrix* is

$$I = I_n = \begin{bmatrix} e_1 & e_2 & \dots & e_n \end{bmatrix}$$

$$:= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (n = 4).$$

The columns of I_n are the *axis vectors* in $\mathbf{R}^n$.

Let

$$p = \begin{bmatrix} p(1) & p(2) & \dots & p(n) \end{bmatrix}$$

be a *permutation* of the integers 1, 2, ..., n. The *permutation matrix* associated with p is

$$P := \begin{bmatrix} e_{p(1)} & e_{p(2)} & \cdots & e_{p(n)} \end{bmatrix}.$$

What is typical of a permutation ~~P~~ p is that *each* integer 1, 2, ..., n occurs *exactly once* among the integers p(1), p(2), ..., p(n). This is like shuffling a deck of cards. After the shuffle *all* the cards remain. They just occur in a different order.

More precisely, a permutation is a *function* from the set {1, 2, ..., n} *onto* the same set. We *present* the function p by listing its function values (as a row):

$$p = \begin{bmatrix} p(1) & p(2) & \dots & p(n) \end{bmatrix}.$$

There are n choices for p(1). After this there are $n-1$ choices for p(2), then $n-2$ for p(3), and so on. So there are n! different permutations of {1, 2, ..., n}.

Example. $n = 3$, $n! = 6$.

$$p_1 = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \leftrightarrow P_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = P_1' = I ,$$

$$p_2 = \begin{bmatrix} 1 & 3 & 2 \end{bmatrix} \leftrightarrow P_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = P_2' ,$$

$$p_3 = \begin{bmatrix} 2 & 1 & 3 \end{bmatrix} \leftrightarrow P_3 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = P_3' ,$$

$$p_4 = \begin{bmatrix} 2 & 3 & 1 \end{bmatrix} \leftrightarrow P_4 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = P_5' ,$$

$$p_5 = \begin{bmatrix} 3 & 1 & 2 \end{bmatrix} \leftrightarrow P_5 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = P_4' ,$$

$$p_6 = \begin{bmatrix} 3 & 2 & 1 \end{bmatrix} \leftrightarrow P_6 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = P_6' =: J .$$

Otherwise put, a permutation matrix P has one 1 in each column and row, and zeros elsewhere. It follows that the transpose P' is also a permutation matrix.

Let

$$A = \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix} \in \mathbb{R}^{m \times n} .$$

Then

$$Ae_i = a_i, \quad i = 1, 2, \dots, n ,$$

and

$$
\begin{aligned}
AP &= A\begin{bmatrix} e_{p(1)} & e_{p(2)} & \cdots & e_{p(n)} \end{bmatrix} \\
&= \begin{bmatrix} Ae_{p(1)} & Ae_{p(2)} & \cdots & Ae_{p(n)} \end{bmatrix} \\
&= \begin{bmatrix} a_{p(1)} & a_{p(2)} & \cdots & a_{p(n)} \end{bmatrix}
\end{aligned}
$$

Postmultiplication by P shuffles the columns according to p. In matlab,

$$
AP = A(:, p) ,
$$

so we have to retain only p, not P!

Let

$$
x = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_n \end{bmatrix} \in \mathbb{R}^n .
$$

Then

$$
e_j' x = \xi_j, \qquad j = 1, 2, \ldots, n ,
$$

and

$$
P'x = \begin{bmatrix} e'_{p(1)} \\ e'_{p(2)} \\ \vdots \\ e'_{p(n)} \end{bmatrix} x = \begin{bmatrix} e'_{p(1)}x \\ e'_{p(2)}x \\ \vdots \\ e'_{p(n)}x \end{bmatrix} = \begin{bmatrix} \xi_{p(1)} \\ \xi_{p(2)} \\ \vdots \\ \xi_{p(n)} \end{bmatrix} .
$$

Premultiplication by P′ shuffles the rows according to p. In matlab,

$$
P'x = x(p) .
$$

Also, if Q is a permutation matrix associated with the permutation q, then

$$
Q'AP = A(q, p) ,
$$

as well as

$$
Q'A = A(q, :) .
$$

We have

$$e_j' e_i = \delta_{ji} := \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} ,$$

the *Kronecker delta*. Thus also

$$e'_{p(j)} e_{p(i)} = \delta_{ji} ,$$

and so

$$P'P = \begin{bmatrix} e'_{p(1)} \\ e'_{p(2)} \\ \vdots \\ e'_{p(n)} \end{bmatrix} \begin{bmatrix} e_{p(1)} & e_{p(2)} & \cdots & e_{p(n)} \end{bmatrix}$$

$$= \begin{bmatrix} e'_{p(j)} e_{p(i)} \end{bmatrix}$$

$$= \begin{bmatrix} \delta_{ji} \end{bmatrix} = I_n .$$

We have

$$e_i e_i' = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (n = 4, i = 2) .$$

In general the *outer product* $e_i e_i'$ has a 1 in its ith diagonal position and zeros elsewhere. Thus

$$PP' = \begin{bmatrix} e_{p(1)} & e_{p(2)} & \cdots & e_{p(n)} \end{bmatrix} \begin{bmatrix} e'_{p(1)} \\ e'_{p(2)} \\ \vdots \\ e'_{p(n)} \end{bmatrix}$$

$$\begin{aligned} &= e_{p(1)} e'_{p(1)} + e_{p(2)} e'_{p(2)} + \ldots + e_{p(n)} e'_{p(n)} \\ &= e_1 e'_1 + e_2 e'_2 + \ldots + e_n e'_n \\ &= I_n \,. \end{aligned}$$

In summary, permutation matrices P satisfy the important relations

$$\boxed{P'P = I_n = PP'} \,.$$

The product of permutation matrices (of the same order n) *is again a permutation matrix* (since a shuffle of a shuffle is still a shuffle!). Let

$$q \leftrightarrow Q = \begin{bmatrix} e_{q(1)} & e_{q(2)} & \cdots & e_{q(n)} \end{bmatrix}$$

be another permutation matrix. To which permutation

$$r = \begin{bmatrix} r(1) & r(2) & \ldots & r(n) \end{bmatrix}$$

does

$$R := PQ$$

correspond? Let

$$A := P$$

so that

$$a_i = e_{p(i)}, \quad 1 \le i \le n \,.$$

Then the ith column of $R = AQ$ is

$$a_{q(i)} = e_{p(q(i))} \,.$$

Thus R corresponds with the *composition*

$$r = p \circ q: \qquad r(i) \equiv p(q(i))$$

of the permutations (functions) p and q.

Example.

$$p = \begin{bmatrix} 2 & 3 & 1 \end{bmatrix} \leftrightarrow P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$

$$q = \begin{bmatrix} 1 & 3 & 2 \end{bmatrix} \leftrightarrow Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

We have

$$\begin{aligned} p(1) &= 2, & q(1) &= 1\,, \\ p(2) &= 3, & q(2) &= 3\,, \\ p(3) &= 1, & q(3) &= 2\,. \end{aligned}$$

Thus

$$\begin{aligned} r(1) &= p(q(1)) = p(1) = 2\,, \\ r(2) &= p(q(2)) = p(3) = 1\,, \\ r(3) &= p(q(3)) = p(2) = 3\,, \end{aligned}$$

that is

$$r = \begin{bmatrix} 2 & 1 & 3 \end{bmatrix}.$$

Check by matrix multiplication:

$$R = PQ = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} e_2 & e_1 & e_3 \end{bmatrix} \leftrightarrow r .$$

In general, if $p \leftrightarrow P$, with what permutation s does $S := P'$ correspond?

The *identity permutation*

$$e := \begin{bmatrix} 1 & 2 & \ldots & n \end{bmatrix},$$

that is

$$e(i) \equiv i \qquad (1 \le i \le n)$$

corresponds with the identity matrix I_n. Since

$$P'P = I_n = PP' ,$$

that is

$$SP = I_n = PS ,$$

we have

$$s \circ p = e = p \circ s$$

that is

$$s(p(i)) \equiv i \equiv p(s(i)) \qquad (1 \le i \le n)$$

Thus s is the *inverse permutation* of p.

Example.

$$p = \begin{bmatrix} 2 & 3 & 1 \end{bmatrix} \leftrightarrow P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

We have

$$p(1) = 2, \qquad p(2) = 3, \qquad p(3) = 1$$

so the inverse permutation s is given by

$$s(1) = 3, \qquad s(2) = 1, \qquad s(3) = 2 .$$

Moreover, the transpose

$$P' = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} e_3 & e_1 & e_2 \end{bmatrix}$$

corresponds with s.

Example.

Let $n = 2m$ be even. The *perfect shuffle* permutation is

$$p := \begin{bmatrix} p(1) & p(m+1) & p(2) & p(m+2) & \dots & p(m) & p(2m) \end{bmatrix}.$$

For instance with $n = 8$,

$$e = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix},$$

$$p = \begin{bmatrix} 1 & 5 & 2 & 6 & 3 & 7 & 4 & 8 \end{bmatrix}.$$

What is the "perfect unshuffle," i.e., the inverse permutation s?

For $n = 8$,

$$s = \begin{bmatrix} 1 & 3 & 5 & 7 & 2 & 4 & 6 & 8 \end{bmatrix}.$$

In general,

$$\begin{aligned} s(i) &= 2i - 1, \quad 1 \le i \le m\,, \\ &= 2i - n, \quad m < i \le n\,. \end{aligned}$$

More on the perfect shuffle.

One of the most important tools in Linear Algebra is the "singular value decomposition" (svd) of a (real or complex) rectangular matrix. The "svd" is the "workhorse" of matlab, and of Applied Linear Algebra. For instance, it will allow us to solve very general linear least squares problems "perfectly." Any *reliable* algorithm for computing an "svd" depends on finding the eigenvalues and eigenvectors of a *Jordan-Lanczos matrix* of the form

$$A = \begin{bmatrix} 0 & B' \\ B & 0 \end{bmatrix}$$

with

$$B = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ & \alpha_2 & \beta_2 & \\ & & \alpha_3 & \beta_3 \\ & & & \alpha_4 \end{bmatrix} \qquad (n = 4)$$

upper *bidiagonal* with positive elements α_k and β_k.

We have

$$A = \begin{array}{cc} \begin{bmatrix} & & & & \alpha_1 & & & \\ & & & & \beta_1 & \alpha_2 & & \\ & & & & & \beta_2 & \alpha_3 & \\ & & & & & & \beta_3 & \alpha_4 \\ \alpha_1 & \beta_1 & & & & & & \\ & \alpha_2 & \beta_2 & & & & & \\ & & \alpha_3 & \beta_3 & & & & \\ & & & \alpha_4 & & & & \end{bmatrix} & \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \\ \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} & \end{array}$$

with the columns and rows in their "natural" order. Now, with P the permutation matrix associated with the perfect shuffle permutation

$$p = \begin{bmatrix} 1 & 5 & 2 & 6 & 3 & 7 & 4 & 8 \end{bmatrix}$$

we have

$$P'AP = \begin{array}{cc} \begin{bmatrix} 0 & \alpha_1 & & & & & & \\ \alpha_1 & 0 & \beta_1 & & & & & \\ & \beta_1 & 0 & \alpha_2 & & & & \\ & & \alpha_2 & 0 & \beta_2 & & & \\ & & & \beta_2 & 0 & \alpha_3 & & \\ & & & & \beta_3 & 0 & \alpha_4 & \\ & & & & & \beta_3 & 0 & \alpha_4 \\ & & & & & & \alpha_4 & 0 \end{bmatrix} & \begin{array}{c} 1 \\ 5 \\ 2 \\ 6 \\ 3 \\ 7 \\ 4 \\ 8 \end{array} \\ \begin{array}{cccccccc} 1 & 5 & 2 & 6 & 3 & 7 & 4 & 8 \end{array} & \end{array}$$

real symmetric tridiagonal with positive next-to-diagonal elements and a *zero main diagonal.* It is not hard to show that the eigenvalues of P′AP, and thus also those of A, occur in $\pm$ pairs. The positive ones are the *singular values* of B. More about singular values, later.

Problem PM1.

Execute the following matlab instructions to verify, experimentally, that what I have said is true, and also to perceive the connections among singular values, 2-norms of matrices, and condition numbers:

```
help mxb
a = rand(4, 1);   b = rand(3, 1);
B = mxb(a, b)
O = zeros(4);   A = [ O   B';   B   O]
p = [1   5   2   6   3   7   4   8];   A(p, p)
lam = eig(A);   mu = eig(A(p, p));
format long, [ lam   mu]
lam = - sort(-lam);
mu = -sort(-mu);
[lam   mu]
lam = lam(1 : 4);   mu = mu(1 : 4);
help svd
s = svd(B);
[lam   mu   s]
help norm
norm(B),   smax = max(s)
1/norm(inv(B)),   smin = min(s)
help cond
cond(B),   smax/smin
```

Problem PM2.

Repeat with B replaced by B : = mxhilbert(10), O adjusted accordingly, and p replaced by p = [1 11 2 12 3 13 ... 10 20] . (Use the ↑ key!)

Remarks on matlab usage.

We have already noted that

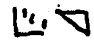
$$A(q, p) = Q'AP ,$$

$$A(:, p) = AP ,$$

$$A(q, :) = Q'P .$$

The latter arises in connection with the partial pivoting strategy which is most frequently used in practice: gfppr, gfpprm and gfpp.

Suppose we have a *full* LU factorization

$$Q'AP = LU =$$ 

of an $n \times n$ matrix A (n pivots). By what we have said above we have the following equivalences:

$$
\begin{aligned}
Ax = b \quad &\Leftrightarrow \quad Q'AP \cdot P'x = Q'b \\
&\Leftrightarrow \quad LU \cdot P'x = Q'b \\
&\Leftrightarrow \quad Lc = Q'b, \quad U \cdot P'x = c.
\end{aligned}
$$

The matlab codes

$$
\begin{aligned}
c &= \mathrm{gfsf}(L, b) \\
x &= \mathrm{gfsb}(U, c)
\end{aligned}
$$

solve the last two systems when $P = Q = I$, and

$$x = \mathrm{gfs}(L, U, b)$$

combines them to solve $LUx = b$. In the general case when we have (nontrivial) permutations p and q, we use

$$
\begin{aligned}
c &= \mathrm{gfsf}(L, b(q)) \ , \\
x(p, :) &= \mathrm{gfsb}(U, c) \ , \\
x(p, :) &= \mathrm{gfs}(L, U, b(q)) \ .
\end{aligned}
$$

We *should* be able to use x(p) in place of x(p, :), but there is a bug in matlab! With partial pivoting (by rows) we have $p = e = \begin{bmatrix} 1 & 2 & \dots & n \end{bmatrix}$. Then we can use x instead of x(p, :).

Reverse order rule for (conjugate) transposition (if the matrices are complex).

We have

$$A'B = \begin{bmatrix} a_j' b_i \end{bmatrix}$$

where, as usual,

$$A =: \left[\, a_1 \;\; a_2 \;\; \dots \;\; a_m \,\right],$$

$$B =: \left[\, b_1 \;\; b_2 \;\; \dots \;\; b_n \,\right].$$

(Conjugate) transpose:

$$(A'B)' = \left[(a_i'b_j)' \right]$$

$$= \left[\, b_j'a_i \,\right] = B'A$$

Now replace A by A′, and so also replace A′ by A, to get

$$\boxed{(AB)' = B'A'}.$$

This uses only the fact that

$$(y'x)' = \overline{\sum_1^n \; \overline{\eta}_j \xi_j}$$

$$= \sum_1^n \; \overline{\xi}_j \eta_j = x'y$$

for (complex) n-vectors x, y.

SUPPLEMENTARY PROBLEMS 1

Key facts:

$$Ax = \begin{bmatrix} a_1 & a_2 & \dots & a_m \end{bmatrix} \begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_m \end{bmatrix}$$

$$= a_1\xi_1 + a_2\xi_2 + \dots + a_m\xi_m$$

is an ℓc (*linear combination*) of the columns of A. Also

$$AB = A \begin{bmatrix} b_1 & b_2 & \dots & b_m \end{bmatrix}$$

$$= \begin{bmatrix} Ab_1 & Ab_2 & \dots & Ab_m \end{bmatrix},$$

provided A has the same number of columns as B has rows. Let

$$A := \begin{bmatrix} 0 & 0 & 3 \\ 1 & -1 & 2 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad B := \begin{bmatrix} 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 0 & 0 & 1 \end{bmatrix},$$

$$C := \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad e := \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix},$$

$$x := \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \quad y := \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}, \quad z := \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

1. Compute the ℓcs $x+y$, $x-y$, $2x$, $3y$, $2x-3y$, $2x-3y+z$.

2. Compute Ae_1, Ae_2 and Ae_3. Compute Be_i, $i=1, 2, 3, 4$.

3. Express x, y and $2x-3y+z$ as ℓcs of e_1, e_2 and e_3.

4. Compute Ax, Ay and Az. Do $A(x+y)=Ax+Ay$ and $A(x-y)=Ax-Ay$? Does $A(2x-3y)=2Ax-3Ay$? Does $A(2x-3y+z)=2Ax-3Ay+Az$?

5. Compute Ab_1, Ab_2 and Ab_3. Thus compute AB.

6. Show that $C=ee^T$ (here we have $e^T := [1 \;\; 1 \;\; 1 \;\; 1]$).

7. Compute BC, A(BC) and (AB)C. Are the last two the same matrix? (Answer: Yes, this is the *associative law* for matrix multiplication). Compute BC as $(Be)e^T$ (that's easier than doing it directly!)

8. Partition

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{11} & A_{22} \end{bmatrix} := \left[\begin{array}{c|cc} 0 & 0 & 3 \\ \hline 1 & -1 & 2 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{array}\right]$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} := \left[\begin{array}{c|ccc} 1 & 0 & 1 & 0 \\ \hline 2 & 1 & 0 & 0 \\ 3 & 0 & 0 & 1 \end{array}\right]$$

Compute $M := AB$ by *block multiplication*. Check with your result from problem 5 above.

9. Compute the outer products xx^T, xy^T, xz^T, yx^T, yy^T, yz^T, zx^T, zy^T, zz^T.

10. Compute the scalar products x^Tx, x^Ty, x^Tz, y^Tx, y^Ty, y^Tz, z^Tx, z^Ty, z^Tz. Compute $\|x\|_2$, $\|y\|_2$ and $\|z\|_2$.

11. Compute A^T, B^T and C^T (note that $C^T=C$, i.e., C is *symmetric*).

12. Compute $A+B^T$. How does this matrix relate with A^T+B?

13. Compute B^TA^T. How does this matrix relate with AB? Compute $C^TB^TA^T$ (parentheses not needed because of the associative law for matrix multiplication). How does this matrix relate with ABC?

14. It is known that, for *real* n-vectors x and y,

$$|y^Tx| \leq \|x\|_2 \; \|y\|_2 \qquad (\textit{Cauchy's inequality}).$$

Thus, for $x \neq 0$ and $y \neq 0$,

$$y^Tx = \|x\|_2 \; \|y\|_2 \cos\theta, \qquad 0 \leq \theta \leq \pi.$$

θ is the (acute) *angle* between the lines, through 0_n, generated by x and y (see the second picture on page AO-7). For the vectors x, y and z above, find the angles between the lines in $\mathbb{R}^3$ generated by:

a) x and y,

b) x and z,

c) y and z.

15. Plot the following vectors in $\mathbb{R}^2$:

$$x_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad x_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \quad x_3 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \quad x_4 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

Which pairs of these vectors are orthogonal?

16. Same question for

$$x_1 = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}, \quad x_2 = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}, \quad x_3 = \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix}, \quad x_4 = \begin{bmatrix} 5 \\ 1 \\ -2 \end{bmatrix},$$

but not in $\mathbb{R}^3$. *Hint: for both problems:* compute X^TX, with

$$X := \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}.$$

17. *Some geometry of LTs* (linear transformations). Consider the following 2×2 matrices:

$$A_1 := \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad A_2 := \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$A_3 := \begin{bmatrix} 2 & 0 \\ 0 & \frac{1}{2} \end{bmatrix}, \qquad A_4 := \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix},$$

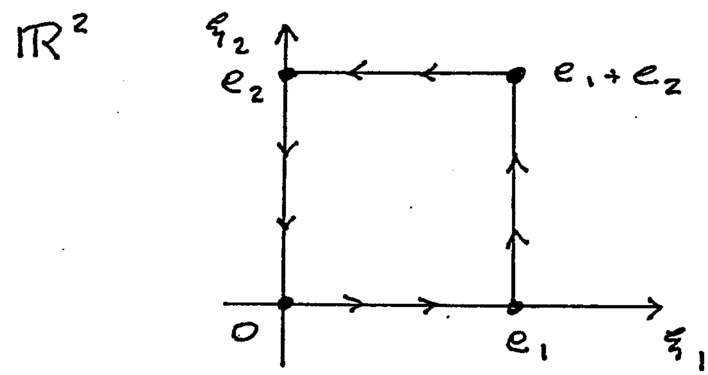
$$A_5 := \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}, \qquad A_6 := \begin{bmatrix} 2 & -1 \\ 1 & 3 \end{bmatrix},$$

$$A_7 := \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}, \qquad A_8 := \begin{bmatrix} 1 & 1 \\ 1 & 1.001 \end{bmatrix}.$$

They represent LTs from $\mathbb{R}^2$ to $\mathbb{R}^2$. For each matrix plot:

a) the image of the *unit circle* $C := \left\{ x \in \mathbb{R}^2 : |\xi_1|^2 + |\xi_2|^2 = 1 \right\}$ under A, starting with $x = e_1 = [0 \;\; 1]^T$ and running *counterclockwise* around the circle. The result will be an ellipse in $\mathbb{R}^2$ with center $0_2 := [0 \;\; 0]^T$. See the m-file ellipse.m in //stewart/gragg/ma1043/mfiles for how to build such a code.

b) The image of the *unit square* S:

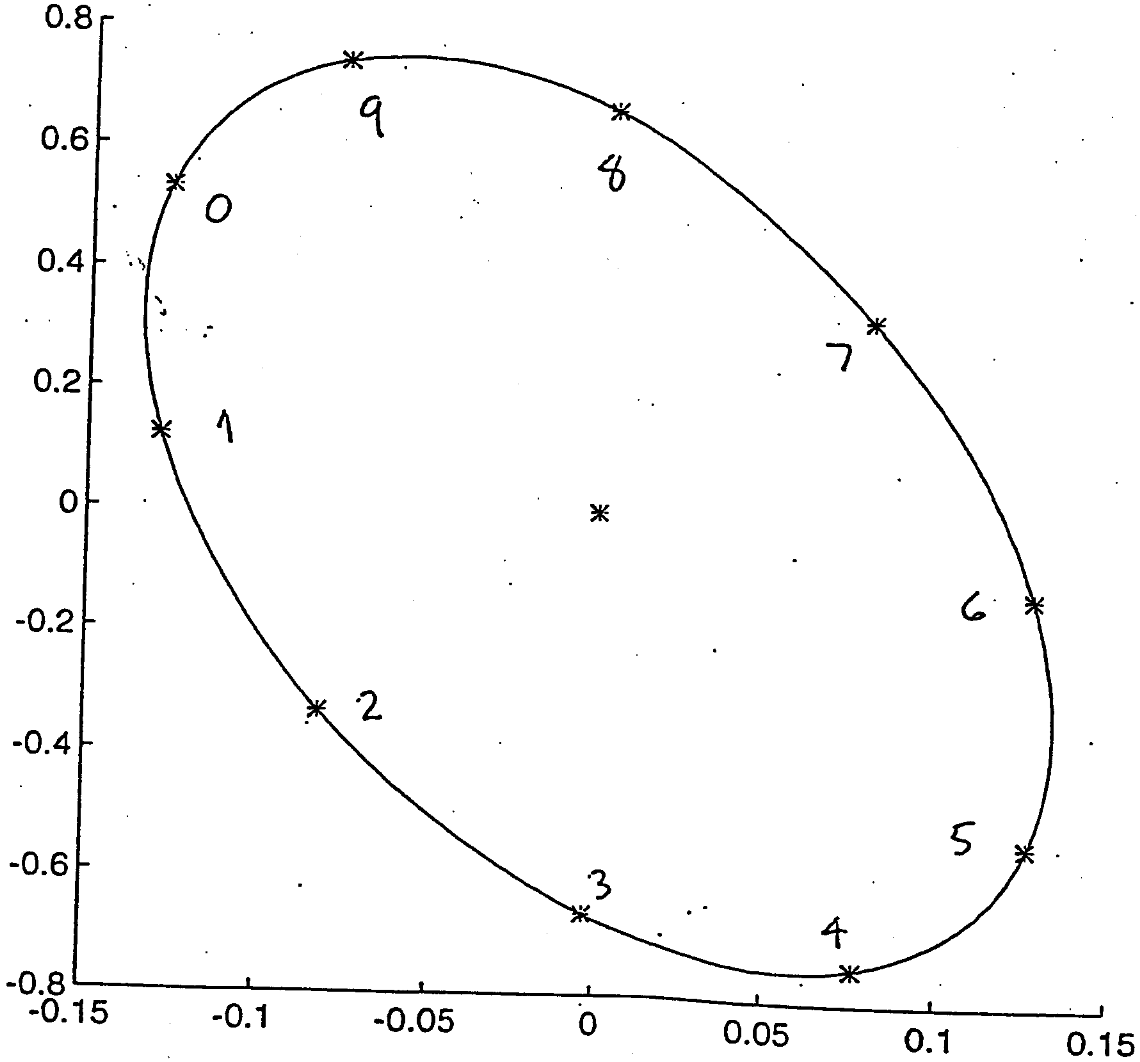


traversed *counterclockwise*, starting from $0_2 = [0 \;\; 0]^T$. Use 81 points on each side of the square, including the corners, but do not repeat the corner points in your list of points. Highlight, for instance as in the code ellipse, every 20th point: 1, 21, 41, 61, 81, 101, ... *Identify* the images under A of the successive points 0, e_1, $e_1 + e_2$ and e_2 on your plots, by hand say. These are just $0 = A0$, $a_1 = Ae_1$, $a_1 + a_2 = A(e_1 + e_2)$ and $a_2 = Ae_2$. Call your code [d A] = parallelogram (A). Here d is the *determinant* of A, d = det (A) in matlab. It is the *area* of the parallelogram $P := AS$. Check this out, roughly, when running your code on the above matrices. More precisely, det A is the *signed* area of P. It is ≥ 0 if P is traversed counterclockwise, ≤ 0 if it is traversed clockwise.

Remarks: A_1 *reflects* points in the line $\xi_1 = \xi_2$. A_2 *reflects* points in the line $\xi_1 = 0$. A_3 *scales* the two variables ξ_1 and ξ_2, but each differently. A_4 *rotates* points through the angle $\frac{\pi}{4}$. A_5 is a "shear transformation," a *unit* upper triangular matrix (ones on the main diagonal). A_8 is (moderately) ill-conditioned.

18. *Experiment* with the codes ellipse and parallelogram, using the ↑ key to execute s = ellipse and d = parallelogram repeatedly. A is *ill-conditioned* if the ellipses (and the parallelograms) are long and narrow. For such matrices s is large and d small. *Read* the code ellipse.

```
function s = ellipse(A)

%       s = ellipse(A):
%
%       If the two by two real matrix A is input this code plots the image in
%       the y-plane of the unit circle in the x-plane generated by the linear
%       transformation y = Ax.  This is an ellipse.  In higher dimensions it
%       is an ellipsoid (like a football in 3D).  If A is not input a random
%       matrix is selected.
%
%       The output s = [s(1) s(2)]' consists of the singular values of A.
%       These are the lengths of the semimajor and semiminor axes of the
%       ellipse, i.e. the half-lengths of the major and minor axes.  You can
%       check this out, roughly, when running the code.  Just type s = ellipse
%       and use the up arrow key to see lotsa ellipse.  When the ellipse are
%       long and skinny the matrix A is ill-conditioned.  This means that
%       cond A := s(1)/s(2) is large.
%
%       One could build a version of this code which does this in 3D, using
%       matlab's graphics.  That would be a very nice project.  Random three
%       by threes would tend to be more ill-conditioned than two by twos.
%       Also, for some reason, random triangular matrices seem to be more
%       ill-conditioned than square ones.  This can be "checked out" by
%       replacing A by A = triu(A) in this code after it is generated.

%       Copyright (c) 4 April 1996 by Bill Gragg.  All rights reserved.

%       ellipse calls no extrinsic functions.


%       begin ellipse

           if nargin < 1

              A = 2*rand(2) - 1;

           else

              [n m] = size(A);

              if m ~= 2 & n ~= 2
                 error('Input matrix not two by two.')
              end

           end

           h = 2*pi/200;    t = 0:h:2*pi;    c = cos(t);    s = sin(t);
           x = [c; s];      y = A*x;

           clg,   axis('square'),   hold on,   plot(y(1,:),y(2,:),'r')

           for k = 1:20:200
              plot(y(1,k), y(2,k),'g*'),   pause(1)
           end

           plot(0,0,'r*'),   s = svd(A);

%       end ellipse
```



$$S_1 = 0.7502$$

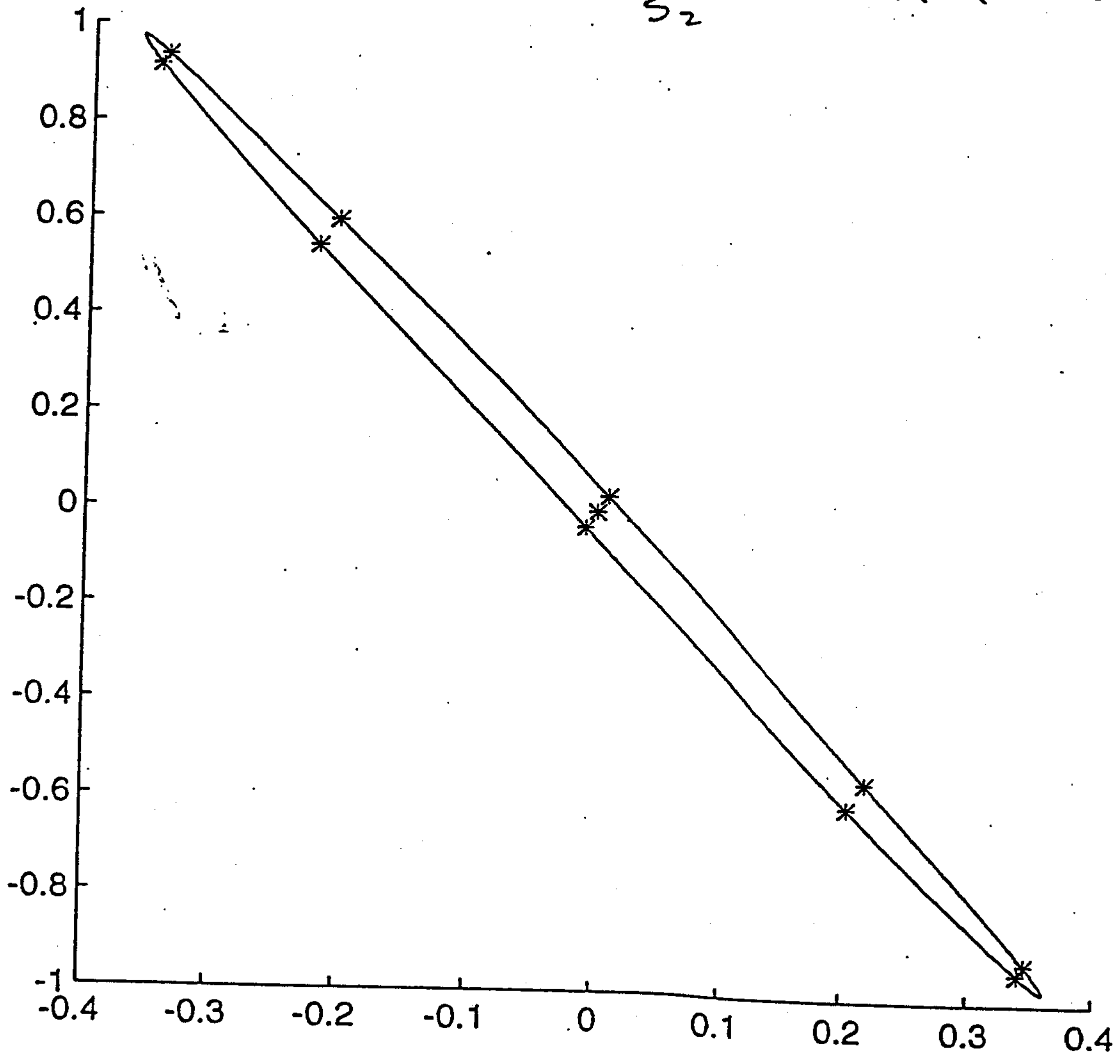
$$S_2 = 0.1206$$

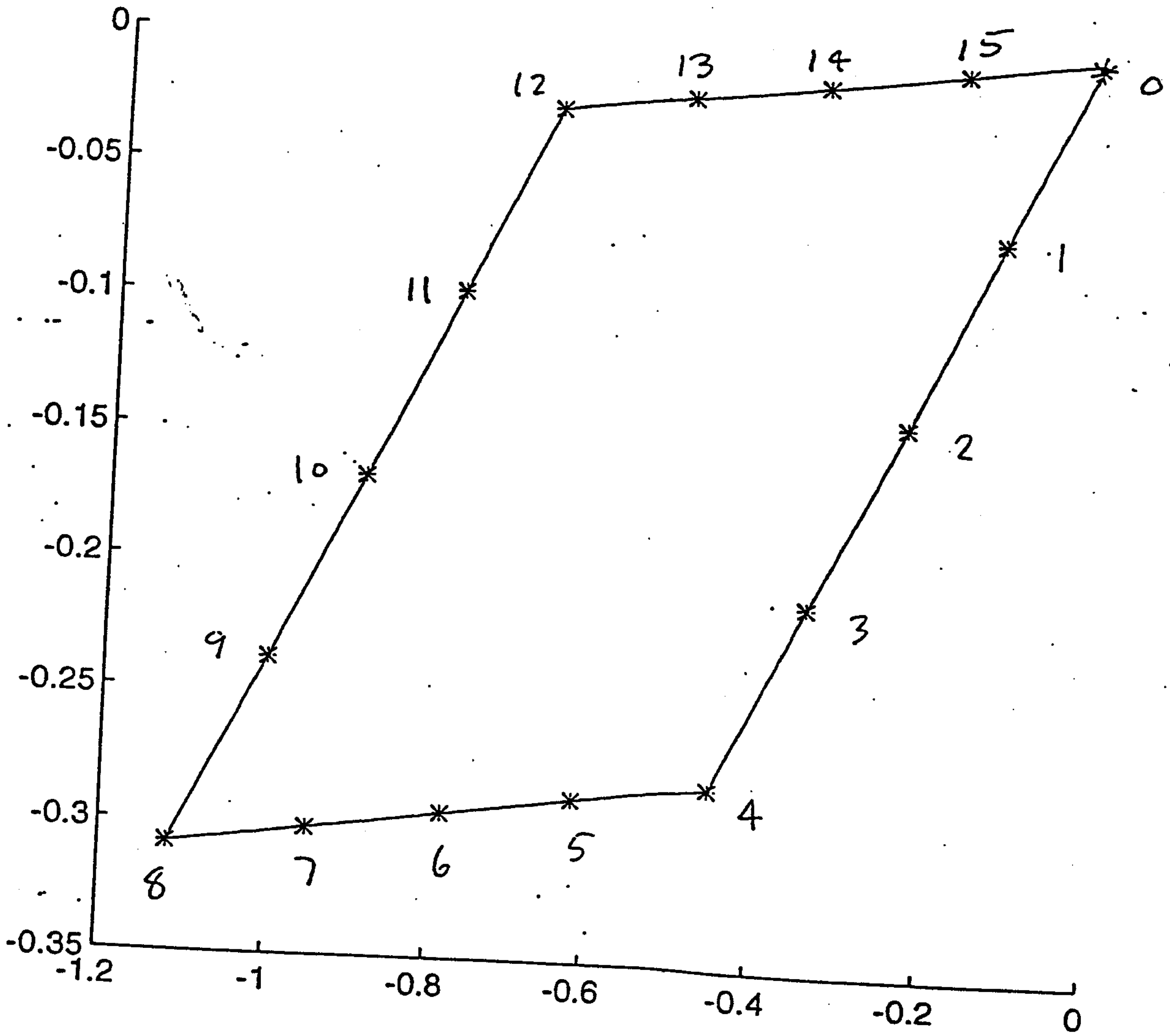
$$C = \frac{S_1}{S_2} = 6.221$$

2 × 2 matrix

$s_1 = 1.0388$
$s_2 = 0.0198$

$c = \frac{s_1}{s_2} = \text{cond}\, A = 52.5$





Clockwise

d = signed area

$= \det A = -0.1756$

```
function x = rotsolve(A,b)

x = rotsolve(A,b):

  This introductory matlab code uses the functions rot and gfsb to SOLVE
the "nonsingular" linear system Ax = b.  We apply ROTATIONS to the
system to transform it to an equivalent upper triangular system Rx = c
and then backsolve this triangular system for x.

Copyright (c) 7 April 1991 by Bill Gragg.  All rights reserved.
Revised 11 July 1997.

rotsolve calls order, rot and gfsb.


begin rotsolve

   n = order(A);    % Gives an error if A is not square.

   Triangularize A.  Backward indexing makes the code elegant.

   for i = 1:n-1

      for j = n-1:-1:i

         Rotate in the "(j,j+1)-plane" to annihilate A(j+1,i).  p and q
         are (row) vectors of indices.  We don't need extra arrays for
         R and c.

         p = i+1:n;           q = j:j+1;              [Q r] = rot(A(q,i));
         A(q,i) = [r; 0];     A(q,p) = Q'*A(q,p);    b(q) = Q'*b(q);

      end

   end

   x = gfsb(A,b);

   I generally don't like to clutter up codes with comments.  If the
   code is well written the comments distract one from seeing the flow
   of the code.  I often make two versions of a code, one with and the
   other without comments.  This code has lots of comments but the
   working part of the code is only 7 lines long.  That's efficiency in
   in terms of "people time" and that's what matlab is all about.

   Of course there is more code in rot and gfsb.  Writing code in this
   modular way aids our understanding of algorithms.  But codes run
   faster if they do not call others.

end rotsolve


Approximate total flops [n = order(A)]:
   Real case:        2n^3/3 adds + 4n^3/3 mults = 2n^3 flops.
   Complex case:     TBC.


Problems.

    Count the real flops in the complex case.  HINT.  A complex add uses
    two real adds and a complex mult, done in the usual way, uses four
    real mults and two real adds.  Thus a complex mult and a complex add,
    done in the usual way, uses four real mults and four real adds.
    However, products of a real and complex number use only two real
    mults.  How much do we save by choosing one of c or s real in rot?
```

2. Prove that this code, with [r; 0] replaced by [r 0]', is NOT correct in the complex case. HINT. Replace [r; 0] by [r 0]' and repeat the diary rotsolve, but now use A = rand(7) + i*rand(7).

function [Q,r] = rot(z)

[Q r] = rot(z):

arefully computes the 2 x 2 (complex) ROTATION Q = [c -s'; s c'] so that Q'z = e_1 r is a scalar multiple of the first axis vector e_1. Rotations are tools of the trade of Computational Linear Algebra.

We have |c|^2 + |s|^2 = 1. The matrix Q is unitary, Q'Q = I = QQ', so

$$z =: \begin{vmatrix} x \\ y \end{vmatrix} = \begin{vmatrix} c & -s' \\ s & c' \end{vmatrix} \begin{vmatrix} r \\ 0 \end{vmatrix} =: QR,$$

a full QR factorization of z, as well as

$$z := \begin{vmatrix} x \\ y \end{vmatrix} = \begin{vmatrix} c \\ s \end{vmatrix} r,$$

a "Gram-Schmidt", or partial, QR factorization of z.

In other words, the vector z is simply scaled to give the unit vector [c; s] (when z\~= 0).

We do not insist that r >= 0, as would be natural in the real case. Instead we take one of c or s >= 0. In particular one of c or s is always real. In the complex case this makes the computation of vectors Q'w faster. For an introductory application see the code rotsolve.

Revised 11 July 1997.

ot calls no extrinsic functions.

```
begin rot

   x = z(1);    y = z(2);

   This computation of r, c and s is used to avoid artificial problems
   caused by underflow and overflow.  For instance the smallest positive
   floating point number is about 5/10^324.  Thus if both x and y were
   1/10^165 then r computed as sqrt(x^2 + y^2) would be zero, even
   though the true value is much larger than the underflow threshhold.

   if abs(x) < abs(y)

      r = x/y;   t = sqrt(1 + r'*r);   c = r/t;   s = 1/t;   r = y*t;

   else

      if x == 0
         Q = eye(2);   r = 0;   return
      end

      r = y/x;   t = sqrt(1 + r'*r);   c = 1/t;   s = r/t;   r = x*t;

   end

   Q = [c -s'; s c'];

end rot
```

Problems.

PROBLEM SET: FORWARD AND BACKWARD SOLUTION OF TRIANGULAR SYSTEMS. MATRIX MULTIPLICATION.

In problems 1–4, solve the given triangular system, $Lc = b$ or $Ux = c$, by hand.

Problem 1.

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & -1 & 1 & 0 \\ -1 & -1 & -1 & 1 \end{bmatrix}, \qquad b = \begin{bmatrix} 2 \\ 1 \\ 0 \\ -2 \end{bmatrix}.$$

Answer: $c = [2,\ 3,\ 5,\ 8]'$.

Problem 2.

$$U = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 8 \end{bmatrix}, \qquad c = \begin{bmatrix} 2 \\ 3 \\ 5 \\ 8 \end{bmatrix}.$$

Answer: $\underline{x} = [1,\ 1,\ 1,\ 1]'$.

Problem 3.

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 3 & 1 & 0 \\ 1 & 7 & 6 & 1 \end{bmatrix}, \qquad b = \begin{bmatrix} 0 \\ -2 \\ -10 \\ -44 \end{bmatrix}.$$

Answer: $c = [0,\ -2,\ -4,\ -6]'$.

Problem 4.

$$U = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 2 & 6 \\ 0 & 0 & 0 & 6 \end{bmatrix}, \qquad b = \begin{bmatrix} 0 \\ -2 \\ -4 \\ -6 \end{bmatrix}.$$

Answer: $x = [1, \ -1, \ 1, \ -1]'$

Problem 5.

Combine the results of problems 1 and 2. Compute $A := LU$ and check that $Ax = b$, by hand.
Answer: $A =$ mxwilkinson(4). See Gauss factorization problem set.

Problem 6.

Combine the results of problems 3 and 4. Compute $A := LU$ and check that $Ax = b$, by hand.
Answer: $A =$ mxvandermonde([1 2 3 4]). See Gauss factorization problem set.

PROBLEM SET: GAUSS FACTORIZATION

Factor the following structured matrices A into A = LU, by hand, using the tableau, and check your work by matrix multiplication. You will better appreciate computers after doing these problems, and they really are interesting! You can find out the complete answers by executing [L U g] = gfpn(A, 0) in matlab. The required special matrix codes are in stewart/home/ma1043/mfiles.
"Do Gauss" – NO PIVOTING. Note well the growth factors g.

Problem 1.

A 4 by 4 matrix due to Wilkinson:

$$W = \text{mxwilkinson}(4),$$

$$W = \begin{bmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 \end{bmatrix}.$$

What is the growth factor for mxwilkinson(n)?

Problem 2.

A Hadamard matrix of order 4:

$$H = \text{mxhadamard}(4),$$

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}.$$

Problem 3.

"Pascal's matrix" of order 5:

$$P = \text{mxpascal}(5),$$

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 6 & 10 & 15 \\ 1 & 4 & 10 & 20 & 35 \\ 1 & 5 & 15 & 35 & 70 \end{bmatrix}.$$

This is an example of a *Cholesky factorization.* We have $U = L$ so that $A = LL'$, with L having *positive diagonal elements.* Also, the *Cholesky factor L* is a part of Pascal's triangle, written another way.

Problem 4.

The (tridiagonal) "negative second difference matrix" of order 5:

$$T = \text{mxtsd}(5),$$

$$T = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}.$$

(The second most important matrix in the Whole Wide World)

Note that the factorization process is very "cheap" for tridiagonal matrices. How cheap!

You can modify the factorization $T = LU$ in a simple way, by an "interior diagonal scaling," to get another Cholesky factorization $T = R'R$, with R upper triangular.

The diagonal elements of U, the pivots, are all positive. Let D be the diagonal matrix formed from their (positive) square roots. Then $T = LDD^{-1}U = R'R$ with $R' := LD$ and $R = D^{-1}U$. Thus one *multiplies* the *columns* of L by the square roots of the pivots and, to adjust for this, *divides* the *rows* of U by these square roots.

Problem 5.

The "negative periodic second difference matrix" of order 5:

$$T = \mathrm{mxpsd}(5),$$

$$T = \begin{bmatrix} 2 & -1 & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ -1 & 0 & 0 & -1 & 2 \end{bmatrix}.$$

The last diagonal element of U will be zero! This can also be modified slightly to be a Cholesky factorization, $T = R'R$, with the last diagonal element of R zero.

Problem 6.

The "min matrix" of order 5:

$$M = \mathrm{mxmin}(5),$$

$$M = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 & 2 \\ 1 & 2 & 3 & 3 & 3 \\ 1 & 2 & 3 & 4 & 4 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}.$$

Another Cholesky factorization!

Problem 7.

The "max matrix" of order 5:

$$M = \mathrm{mxmax}(5),$$

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 2 & 3 & 4 & 5 \\ 3 & 3 & 3 & 4 & 5 \\ 4 & 4 & 4 & 4 & 5 \\ 5 & 5 & 5 & 5 & 5 \end{bmatrix}.$$

Problem 8.

A 4 by 4 Vandermonde matrix built from the "abscissas" 1, 2, 3, 4.

$$V = \text{mxvandermonde}([1\ \ 2\ \ 3\ \ 4]),$$

$$V = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \end{bmatrix} \quad \text{(V is of } \textit{theoretical} \text{ interest)}$$

Problem 9.

The 3 by 3 Hilbert matrix:

$$H = \text{rats}(\text{mxhilbert}(3)),$$

$$H = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}.$$

Problem 10.

The "idft matrix" of order 4 (the *most important* matrix in the Whole Wide World):

$$W = \text{mxidft}(4),$$

$$W = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \quad (i := \text{sqrt}(-1))$$

Complex matrices rarely arise in practice but, when they do, they seem to be important. mxidft(n) is probably the most important matrix of all time, the matrix used in the *fast Fourier transform*. Typical orders are $n = 1024$ and $n = 4096$!. Here we use the case $n = 4$ as a, fairly massive (!), drill in complex arithmetic.

Complete pivoting for size or, more precisely, pivoting to *prevent growth.*

This kind of pivoting should *not* be confused with the term "pivoting" that is used in the field of linear constrained optimization (linear programming), nor should the term "programming" which is used in that field be confused with the programming of computers! Confusing, isn't it?

Problem 11.

Factor the Wilkinson matrix of problem 1 using complete pivoting: $Q'AP = LU$. What is the growth factor now? Answer: $g = 2$!

Problem 12.

Factor $Q'AP = LU$ for the following matrix A in *three* ways, using no pivoting, complete pivoting, and any *other* pivot scheme you choose.

$$A = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 2 & 1 \\ 1 & 3 & 3 \\ 1 & 4 & 5 \end{bmatrix}.$$

Pivots are, by definition, not zero. How many pivots are there in *each* case? We will ultimately show that, for a given matrix A, *every* pivot scheme will always find the *same* number of pivots, and this no matter how large the matrix A!

These are examples of the

LU Theorem. Let A be n by m with $A \neq O$. There are permutation matrices P and Q, and an integer r with $1 <= r <= \min(m, n)$, so that

$$Q'AP = LU ,$$

with L unit lower trapezoidal with r columns, and U upper trapezoidal with nonzero diagonal elements (and r rows).

Now you have "paid your dues" so you can use matlab.

Problem 13.

Factor the 8 by 8 versions of the matrices in problems 1–11 with matlab. Use all three codes: gfpn, gfppr and gfpc. *Do not output the factorizations*! Compare the growth factors g obtained by these three pivot strategies. Check the factorizations by displaying the respective scaled errors

$$\text{en} = \text{norm}\,(\text{A} - \text{L}*\text{U})/\text{a}$$
$$\text{ep} = \text{norm}\,(\text{A}(\text{q}, :) - \text{L}*\text{U})/\text{a}$$
$$\text{ec} = \text{norm}\,(\text{A}(\text{q}, \text{p}) - \text{L}*\text{U})/\text{a}$$

where

$$\text{a} = \text{norm}\,(\text{A})\,.$$

Simultaneously, record the *condition numbers*, cond A, of these matrices. For instance the "one liner" cond(mxhilbert(8)) gives the condition number of the 8 by 8 Hilbert matrix.

Repeat with $n = 8$ replaced by $n = 16$, or do these simultaneously.

(Not so) roughly speaking one can expect to lose log10(condA) decimal digits of accuracy when solving $Ax = b$ for x on a computer, for square matrices A. We always have cond A >= 1. A is *ill-conditioned* if cond A is *large*. One might think that large growth and ill-conditioning are related. The following example shows that this is not true.

We have $g = 2^{n-1}$ for W = mxwilkinson(n) and partial pivoting, but $g = 2$ if complete pivoting is used, it appears. (One can prove this!) What is cond(mxwilkinson(200))? How long does it take matlab to compute it? How about $n = 500$?

Problem 14.

Factor the matrix in problem 12 by using matlab and gfpc, and check that ec is of the order of magnitude of the machine precision eps.

Problem 15.

Execute "eps" and "binrep(eps)", or just "br(eps)". Then execute "eps = machprec([illegible])", to replace eps by its "correct" value. We won't have to know the fine details concerning eps.